多値論理研究ノート　第 38 巻　第 7 号　2015 年 9 月
Note on Multiple-Valued Logic in Japan　Vol.38, No.7 Month 2015

２乗則を用いたデジタル乗算器アルゴリズムの検討

Study of Digital Multiplier Algorithm Using Addition and Square Formula

佐々木秀　小林春夫
Shu Sasaki　Haruo Kobayashi
群馬大学大学院　理工学府 電子情報部門
Division of Electronics and Informatics, Faculty of Science and Technology, Gunma University
1-5-1 Tenjin-cho Kiryu Gunma 376-8515 Japan
Phone: 81-277-30-1789　Fax: 0277-30-1707
e-mail:　t15804040@gunma-u.ac.jp　k_haruo@el.gunma-u.ac.jp

概要　デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられているが、直接的に実現すると全加算器の２次元配列になり、回路規模・消費電力・演算時間が比較的大きくなってしまう。そのため回路規模・消費電力の減少および高速化のために様々なアルゴリズムが提案され、それに基づきデジタル乗算器が設計・実現されてきている。この論文では入力 A, B に対し　その和と２乗から積 AB を計算する式を用いてデジタル乗算器を設計する方式を検討した。また、そこで A, B のそれぞれの上位ビット、下位ビットに分割して計算量を低減する方式（Divide & Conquer）を用いる。提案アルゴリズムを FPGA で実現・検証をすべく、検討したアルゴリズムで 8bit×8bit （出力：16bit）, 16bitx16bit（出力 32bit)の乗算器を設計し RTL シミュレーションを行い、正しくデジタル乗算が行われていることを確認した。提案方式で内部をパイプライン構成にすれば高速化が可能であり、小規模・高速のデジタル乗算器を実現でき、一つのチップ内に多数配置でき大量のデータ処理が可能になる。

1　研究背景と目的

デジタル乗算器はデジタル計算機、DSP チップ等に広く用いられている。乗算は加算を何回も繰り返して計算できる、すなわち多くの計算量を必要とする。したがって乗算器をそのまま実現すると必然的に全加算器の２次元配列になり、回路規模・消費電力・演算時間が比較的大きくなってしまう。

そのため回路量・消費電力の減少および高速化のために様々なアルゴリズムが提案され、それに基づきデジタル乗算器が設計・実現されてきた。特にデジタル通信システムでは複雑・大量のデジタル演算をリアルタイムに行う必要があり、小規模なデジタル乗算器を実現できれば、それを多数実装できるので、そのアプリケーションにはきわめて有効である。

ここでは２つのデジタル入力 A, B に対し　その和と２乗から積 AB を計算する下記の式を用いてデジタル乗算器を実現する方式を検討する。

$$AB = \frac{1}{2}\{(A + B)^2 - A^2 - B^2\} \quad \cdots\cdots\cdots(1)$$

この２乗演算のためにルックアップテーブル（Look-Up Table: LUT）を用いる。また、LUT のサイズを小さくするため A, B, A+B のそれぞれの上位ビット、下位ビットに分割して計算量を低減する方式（Divide & Conquer）を用いる。

提案アルゴリズムを FPGA で実現・検証をすべく、検討したアルゴリズムで 8bit×8bit （出力は

16bit)、16bit x 16bit (出力 32bit) の乗算器を設計し RTL（Register Transfer Level）シミュレーションを行い、正しくデジタル乗算が行われていることを確認する。提案方式で内部をパイプライン構成にすれば高速化が可能であり、小規模・高速のデジタル乗算器を実現でき、一つのチップ内に多数配置でき大量のデータ処理を行うことができる。なお、ここでは簡単のため A, B は正の数のみを扱っている。

2　検討するデジタル乗算器アルゴリズム

2.1 検討アルゴリズム

検討したアルゴリズムは前述で示した式（1）を使い 1/2 は右シフト演算（実際には結線変更のみでよい)、二乗の計算部分を LUT を用いることで回路規模削減・低消費電力化・計算時間短縮を実現する。

図 1 に式(1)を実現する回路構成を示す。

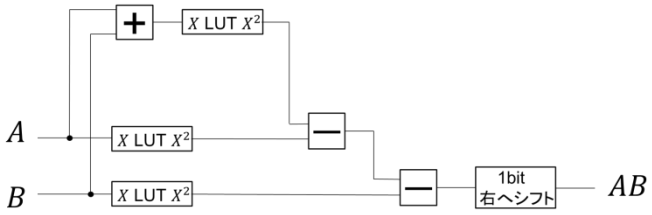

図 1：式(1) を実現する回路構成

LUT はメモリ(RAM または ROM)である。入力はメモリの”アドレス”であり、出力はメモリの”データ”である。メモリ内に計算データを記憶しておくことで、LUT で任意の計算結果を得ることができる。

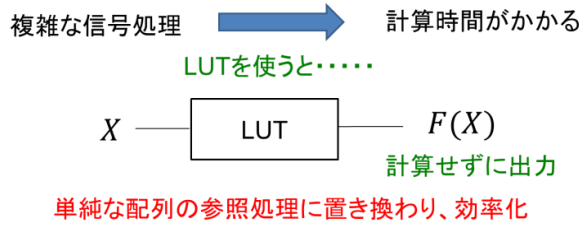

図 2：ルックアップテーブル(LUT)回路

図 1 では３つの LUT を用いている。他の構成として演算時間は約３倍と大きくなってしまうが回路量を減らすためたとえば１つの LUT を A^2, B^2, (A+B)^2 の計算を行うために逐次的に使うこともできる。この際３つの加減算器も１つにすることができ、全体として３分の１の回路量でよい。

2.2 対数を用いるアルゴリズムとの比較

乗算を変換する方法の一つとして、対数を用いて演算する手法も考えられる。すなわち２つのデータ A, B に対して AB の乗算を、加算器と LUT を用いて次のように計算する。

① A, B の対数 log A, log B を
　対数データの LUT を用いて得る。
② log A + log B (= log AB) を加算器で得る。
③ log AB から AB を
　指数データの LUT を用いて得る。

しかしながら高精度で対数、指数を得るためには LUT のビット数を大きくしなければならず、回路規模が大きくなってしまうため、今回は考えないものとする。

2.3 将来の検討アルゴリズム

次の計算アルゴリズムも有力である。

$$AB = \frac{1}{4}\{(A + B)^2 - (A - B)^2\} \quad \cdots\cdots\cdots(2)$$

加減算回数は３回と同じであるが、LUT 参照処理は２回になる。だがここでは簡単のため A, B が正の数の場合のみを扱っているので　式(1)を検討した。次のステップとして式(2)を検討する予定である。

2.4　Divide & Conquer について

ここでは二乗計算を担う LUT の回路量低減について検討する。

適当な入力値 A のビット数 n を大きくすると出力A^2の 計算精度を保つためには LUT のサイズを（n^2に比例して）大きくしなければならない。そ

こで n を上位ビットと下位ビットに分割して計算させることを考える。

たとえば図 3 のように入力 A が 8 ビットで、その上位 4bit をA_H, 下位 4 ビットをA_L とする。ここでA^2の計算は次のようになる。

$A^2 = A_H^2$ (8bit 左シフト) $+ 2A_HA_L$ (4bit 左シフト) $+ A_L^2$ ……(3)

式(2) を用いた入力 A (8bit) の 2 乗A^2 (16bit) 回路の計算回路を図 3 に示す。8 ビットの入力信号 A を上部 4 ビットと下部 4 ビットに分割してビット数を減らしてから計算し、シフト演算によって桁を揃える。

例：8bit(C=11001001)について考える。

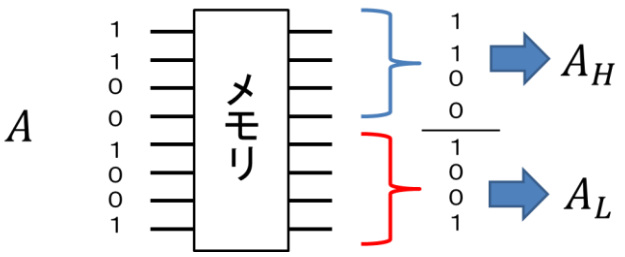

図 3：データ A の上位、下位ビット分割

分割したビットはさらに分割することも出来る（Divide & Conquer）。このようにビット数を分割することによって小さなビット数の LUT を用いて演算することができる。

図 4 にこの方式を用いた入力（8bit×8bit）で出力 16bit の乗算回路構成を示す。

図 1 の二乗部分を図 5 の回路構成で実現している。このとき使用する LUT のビット数は 8bit から 4bit に下がっている。

同じ手法を繰り返し行うことで、大きなビット数の計算を小さなビット数の LUT で計算することができる。

図 5 を参照し、8bit で A = 11001001= $(201)_{10}$について考える。

まず 10 進数を用いて理論計算を行う。

上部（A_H）4bit、下部（A_L）4bit に分割すると

$A_H = 1100\ (12)_{10}$、

$A_L = 1001\ (9)_{10}$

となり、これを使って演算を行っていく。

まず分割したA_H、A_Lを 2 乗する。

$A_H^2 = 12^2 = 144$

$A_L^2 = 81$

次にA_H^2を 8bit 左にシフトして桁を合わせる。

A_H^2 (8bit 左) $= 12^2 \times 256 = 36864$

次に分割したA_H、A_Lを足し、2 乗する

$A_H + A_L = 12 + 9 = 21$

$(A_H + A_L)^2 = 21^2 = 441$

そして求めた値を回路に基づいて減算していく。

$(A_H + A_L)^2 - A_H^2 = 441 - 144 = 297$

$(A_H + A_L)^2 - A_H^2 - A_L^2 = 297 - 81 = 216$

$\{(A_H + A_L)^2 - A_H^2 - A_L^2\}$ (4bit 左)

$= 216 \times 16 = 3456$

最後に A の 2 乗と、分割して求めた解が合っているか確かめる。

$(A_H)^2$ (8bit 左) $+ (A_H + A_L)^2$ (4bit 左) $+ (A_L)^2$

$= 36864 + 3456 + 81$

$= 40401 = 201^2 = A^2$

となり、分割して求めた解とA^2の値が一致したので、この回路の正当性が示せた。

次に、2 進数において上記と同様の手順で理論計算を行う。

上部（A_H）4bit、下部（A_L）4bit に分割すると

$A_H = 1100\ (12)_{10}$、

$A_L = 1001\ (9)_{10}$

となる。

$A_H^2 = 10010000\ \ (144)_{10}$

A_H^2 (8bit 左) $= 1001000000000000\ \ (36864)_{10}$

$A_L^2 = 1010001\ \ (81)_{10}$

$A_H + A_L = 00010101\ \ (21)_{10}$

$(A_H + A_L)^2 = 110111001\ \ (441)_{10}$

$(A_H + A_L)^2 - A_H^2 = 100101001\ \ (297)_{10}$

$(A_H + A_L)^2 - A_H^2 - A_L^2 = 11011000\ \ (216)_{10}$

$\{(A_H + A_L)^2 - A_H^2 - A_L^2\}$ (4bit 左)

$= 110110000000 \quad (3456)_{10}$

$(A_H)^2$ (8bit 左) $+ (A_H + A_L)^2$ (4bit 左) $+ (A_L)^2$

$= 1001000000000000 \quad (36864)_{10}$

$+ 110110000000 \quad (7056)_{10}$

$+ 1010001 \quad (81)_{10}$

$= 1001110111010001 \quad (40401)_{10} = A^2$

となり、10 進数の場合と同じ出力結果を得られることが分かり、先ほどと同様であることが示せた。

2.5 問題の分割（Divide & Conquer） による演算速度の向上

大きなビット数を持つ LUT を分割することで回路面積を小さくすることが出来ると同時に、演算速度も大幅に縮小することが出来る。

参考文献[3]より、加算器はビット数が大きくなるほど消費電力や計算時間を大きくなってしまう。そのためデータを分割することによって、演算処理をする加算器側のビット数が減り、演算速度が向上させる。また、より大きなビット数から分割することによって、その向上性能も大きくなることが分かる。

演算の際に生じる遅延時間もビット bit 数が小さいほど少ないことが分かっている。これによって遅延による誤差も減少させることができる。

ビット数が半分になると、面積、演算速度、消費電力も半分になるため、分割して計算することで性能が向上する。ここでは簡単のため、4bit まで分割して理論計算と回路構成を行った。

3 シミュレーションによる検証

検討アルゴリズム・回路構成の正当性、つまり検討した内容が実装に反映させることが出来るかどうかを検証するため「Verilog HDL」を用いて回路シミュレーションを行った。

具体的には、シミュレーションソフト上に図 4 の回路構成を実現し、二つの入力値を変えて演算させて結果を出力し、結果が正しいかどうかをチェックした。入力が 4bit×4bit で出力が 8bit の場合は、数字の組み合わせが 16×16 = 256 通り、入力が 8bit×8bit で出力が 16bit の場合、256×256 = 65536 通り、入力が 16bit×16bit で出力が 32bit の場合は 65536×65536 = 4294967296 通りあるが、それらすべての数値で提案アルゴリズムが正しく乗算を行っていることを確認した。

このプログラムを使用すれば提案アルゴリズムの FPGA 実現ができる。

4 まとめと今後の課題

二乗則の計算アルゴリズムを用いた回路構成と、そのハードウェア実現法を検討した。今後は実際に FPGA 実装を行い提案構成の実機検証していく。具体的には次を重点的に検証していく。

・計算精度と演算器・LUT のビット数の明確化
・実装 FPGA 動作クロック周波数）と
　計算速度の明確化
・マイナスの数値の場合の計算。
　デジタル乗算器はすべて２進数で表現されており、マイナスがつくと２の補数によってマイナスを表現する。ビット分割の際にはその配慮が必要であるが、今後はそれに対応していく。
・式(2) のアルゴリズムの適用

参考文献

[1] A. V. Oppenheim, and R. W. Shafer, Digital Signal Processing, Printice-Hall (1975)

[2] L. Cohen, Time-Frequency Analysis, Prentice Hall

[3] A Technical Tutorial on Digital Signal Synthesis, Analog Devices, Inc. (1999).

[3] N. Weste, D. Harris, CMOS VLSI Design: A Circuits and Systems Perspective, Addison Wesley (2010)

[4] 佐々木秀、小林春夫「短時間スペクトラム解析の計算アーキテクチャの検討」第 5 回電気学会東京支部栃木・群馬支所合同研究発表会 ETT-15-69, ETG-15-69 宇都宮 (2015 年 3 月)

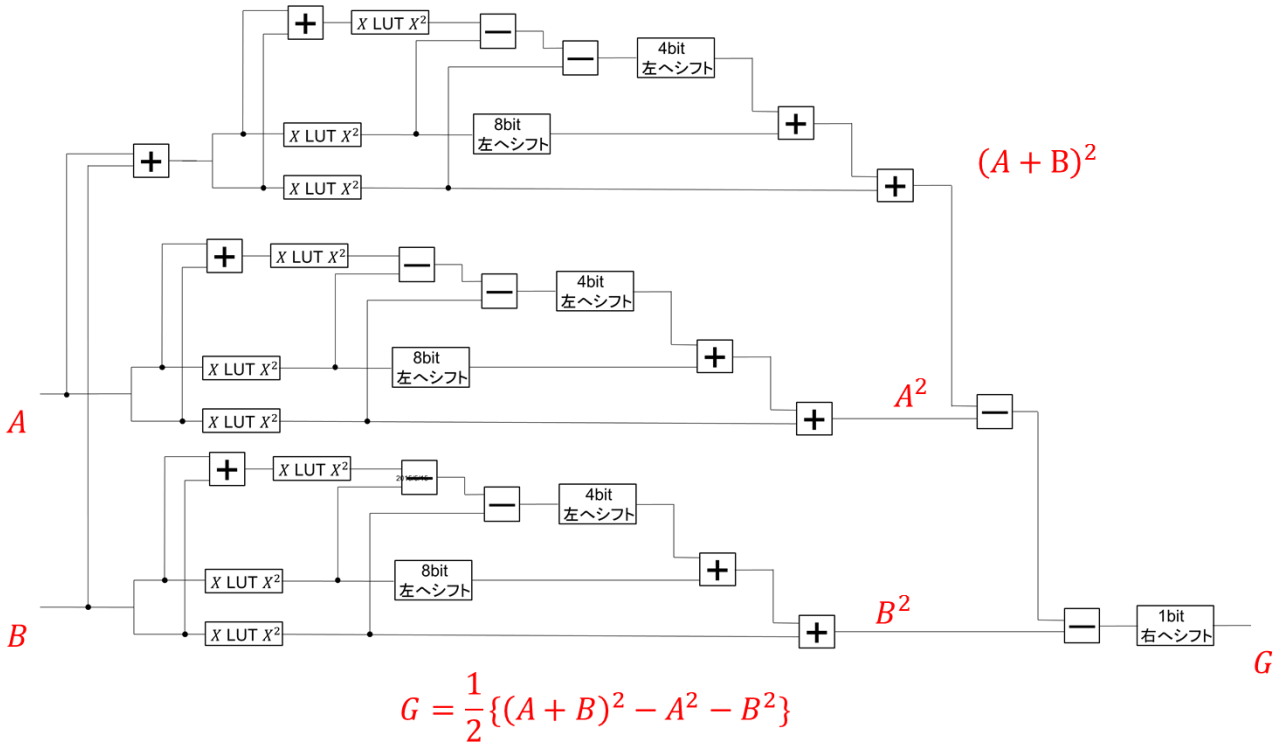

図 4：8bit×8bit の回路構成

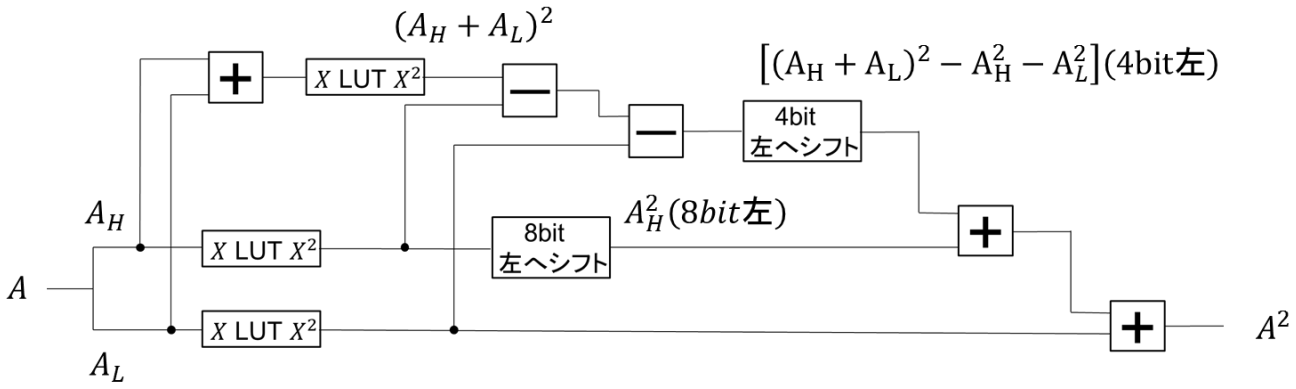

図 5：入力 A(8bit)の 2 乗A^2 (16bit)の計算回路

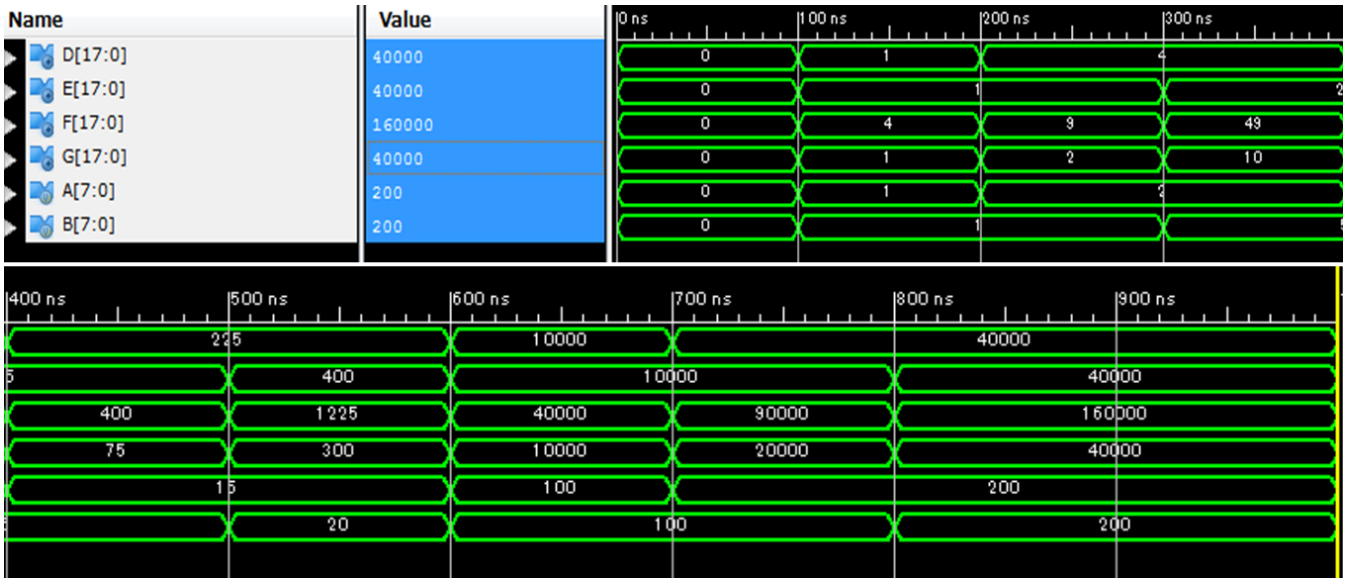

図 6：8bit×8bit のシミュレーション

ここでは図 4 の回路プログラムを入力し、入力値 A , B の値を 100[ns]、または 200[ns]毎に変化させ、その区間における演算結果を波形上に表示している。

演算結果の確認として

D（18bit） ＝ A^2

E（18bit） ＝ B^2

F（18bit） ＝ $(A + B)^2$

を表示しており、

これらを用いて

$$G（18bit） = \frac{1}{2}\{(A+B)^2 - A^2 - B^2\}$$

を演算している。

図 6 中の D , E , F , G それぞれには処理することのできる最大 bit 数を示している。この bit 数は入力の bit 数によって変化し、入力値の bit 数の倍以上の容量が必要になる。

Value における数値はシミュレーションの最終結果を表示しており、今回のシミュレーション結果での最終値は

A = 200

B = 200

D = 40000

E = 40000

F = 160000

G = 40000

となるため、Value ではその値が表示されている。

また、これらの計算はすべて２進数で行っているが、簡単のため結果の表示を１０進数にしている。

図 6 中に示されたとおり、それぞれの入力値 A , B に対して出力 G は A×B の値を導いている。これは検討したアルゴリズムが回路上に反映できるということが示せた。